Due Date: Friday 13th Sept. 2013 at 17:00 (week 07).

You are required to attempt all the questions listed below, and submit your work in a single .zip file containing document (in either MS word or PDF format) and java source code.

The indicative marking scheme is shown below.

Question 1: [60 marks, all subquestions have equal marks]

You are required to calculate the time complexity using the big-O notation of the *myAlgorithm(int n)* method defined by the following Java code.

Code: Algorithm_1

```
void algorithm_1(int n) {
  if (n < 1) return;
  System.out.println(q(1, n)*n);
  System.out.println(r(n));
  System.out.println(q(1, n+n) + r(n+n));
}
```

```
int q(int i, int n) {
  return i+(i >= n ? 0 : q(i+i, n));
}
```

```
int r(int n) {
  int sum = 0;
  for (int i=1; i <= n+n; i++)
    sum+=i + q(1,n);
  return sum;
}
```

```
int t(int n) {
  for (int i=1; i <= n+n; i++) {
for (int j=1; j < i; j++)
    sum+=i + q(1,n);
  return sum;
}
```

1. For $n > 0$, what is the time complexity of the method *q(1, n)*. Show the details of your calculation of *O(q(1, n) ≈ O(?).*
2. For $n > 0$, what is the time complexity of the *r(n)* method. Show the details of you calculation of *O(r(n)) ≈ O(?).*
3. For $n > 0$, what is the time complexity of the *algorithm_1(int n)* method. Show the details of you calculation of *O(algorithm_1(n)) ≈ O(?).*

Question 2: [40 marks]

You are required to calculate the time complexity using the big-O notation of the *Algorithm_2* given by the following methods defined by the following Java code.

1. Explain how the *binarySearch(array[n], key)* algorithm works support your answer with an illustration of this search algorithm.

[15 marks]

2. For $n > 0$, what is the time complexity of the *binarySearch(array[n], key)* algorithm. Show the details of you calculation of *O(binarySearch(array[n], key)) ≈ O(?).*

[15 marks]

Code: Algorithm_2

```
int binarySearch(int[] array, int key) {
        int lo = 0, mid, hi = array.length-1;
        while (lo <= hi) {
            mid = (lo + hi)/2;
            if (key < array[mid])
                hi = mid - 1;
            else if (array[mid] < key)
                lo = mid + 1;
            else return mid; // success
        }
        return -1;      // failure
}
```

3. Write a Java program that counts the number of operations the *binarySearch* algorithm executes to search a given array of size *n*. Hint: you can simply extend the above code say the *while loop* to count the number of iterations it executes for a given array size *n*. The program should output say two variables *n* and *iterated* respectively for the array size and the number of iterations the *binarySearch* has executed.

Assuming, that *eachAlgorithm_2* iteration takes a constant $timet0.01$ s (seconds), plot the experimental time complexity of *binarySearch(n),* which can be calculated as: *t(n)iterated*t.*

[10 marks]